VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Aplikace pro identifikaci obsazenosti parkoviště

Parking Lot Occupation Identification Application

2013 Richter Lukáš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Richter**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Aplikace pro identifikaci obsazenosti parkoviště
Parking Lot Occupation Identification Application

Zásady pro vypracování:

Při vývoji aplikací digitálního zpracování obrazu je zapotřebí ověřit výsledky algoritmů proti ručně získaným hodnotám. Jednou z takových aplikací může být detekce obsazenosti parkovišť. Cílem práce je vytvoření aplikace, která bude umožňovat editaci obsazenosti parkovišť z pořízeného video záznamu formou webové aplikace. Dále bude aplikace schopna ukládat výsledky algoritmů pro konkrétní snímky z videí a porovnávat je s ručně získanými hodnotami. Ve své práci proveďte:

1. Seznamte se s frameworkem Django pro vývoj webových aplikací,
2. vytvořte webovou aplikaci pro editaci obsazenosti parkovišť, která bude ukládat data na server,
3. naprogramujte aplikaci pro příjem dat z algoritmů zpracování obrazu, krerá bude tato data porovnávat s ručně získanými daty,
4. svou aplikaci řádně otestujte a zdokumentujte.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Gaura**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 5. května 2013 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2013 .....

Rád bych na tomto místě poděkoval Ing. Janu Gaurovi za odborné vedení během mé práce.

Abstrakt

Náplní této práce je návrh a implemntace aplikace pro editaci obsazenosti parkoviště z pořízeného videozáznamu a aplikace pro příjem a vyhodnocení dat z algoritmů zpracování obrazu. Aplikace jsou vyvíjeny formou webových aplikací. Tyto aplikace umožní otestovat výsledky algoritmů pro digitální analýzu a zpracování obrazu proti ručně zadaným hodnotám.

Klíčová slova: Python, Django, webová aplikace, AJAX, JavaScript

Abstract

Purpose of this project is to design and implement application for editing parking lot occupation from recorded video and another application which receives and evaluates data from the image processing algorithms. Applications are developed as web applications. These applications allow to test results of algorithms for digital image processing and analysis against a manually enterd values.

Keywords: Python, Django, web application, AJAX, JavaScript

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
BSD	–	Berkeley Software Distribution
CD	–	Compact Disc
CSS	–	Cascading Style Sheets
CRUD	–	Create-Read-Update-Delete
CSV	–	Comma-separated values
DRY	–	Don't Repeat Yourself
HTML	–	HyperText Markup Language
HTTP	–	Hypertext Transport Protocol
MVC	–	Model View Controller
MTV	–	Model Template View
ORM	–	Object Relational Mapping
PIL	–	Python Image Library
URL	–	Unique Resource Locator
XML	–	Extensible Markup Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Digitální zpracování a analýza obrazu jsou vědecké disciplíny, jejichž výsledky jsou dnes již běžně využívané, at' už při úpravách a kompresi obrazu nebo ve složitých detekčních systémech.

V současné době existuje mnoho algoritmů zabývajících se digitálním zpracováním a analýzou obrazu. Z tohoto důvodů je nutné vyvíjet takové aplikace, které by umožňovaly tyto algoritmy otestovat a vzájemně porovnat. Tato práce se zabývá návrhem a implementací právě takovéto aplikace, konkrétně aplikací pro identifikaci obsazenosti parkoviště.

Bakalářská práce je rozdělena do tří kapitol. První kapitola má čtenáři přiblížit, co je cílem této práce. Dále se v této kapitole nalézají reference na jiná podobná řešení.Ve druhé kapitole jsou shrnuty nejdůležitější technologie použité při vývoji této práce. Tato kapitola seznámí čtenáře se základními informacemi a vlastnostmi jednotlivých technologií. Poslední kapitola se zabývá jednotlivými fázemi vývoje aplikace. Tyto fáze jsou rozděleny pomocí podkapitol na analýzu, návrh, implementaci a testování.

2 Cíl práce

Cílem této práce bylo seznámení se s problematikou tvorby webových aplikací pomocí Django Frameworku a vytvořit webovou aplikaci, která umožňuje editaci obsazenosti parkoviště z pořízeného videozáznamu. Tato aplikace najde uplatnění při vývoji algoritmů pro digitální zpracování obrazu, hlavně v oblasti testování úspěšnosti těchto algoritmů. Dále bylo nutno vytvořit aplikaci umožňující nahrát výstupy testovaných algoritmů (ve formátu XML nebo CSV) na server, kde se porovnají hodnoty získané z nahraného souboru s hodnotami zadanými ručně pomocí aplikace pro editaci obsazenosti parkoviště. Výsledek porovnání se uloží do databáze a zobrazí v přehledné tabulce.

2.1 Middlebury Stereo Evaluation

Podobné řešení vzniklo na univerzitě Middlebury jako součást projektu „A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms“. Toto řešení umožňuje porovnávat výkon algoritmů pro zpracovávání trojrozměrného obrazu. Řešení je přístupné přes internet na adrese „*http://vision.middlebury.edu/stereo/*“. [1]

2.2 The KITTI Vision Benchmark Suite

Další podobné řešení patřící k projektu KITTI se zabývá porovnáním algoritmů pro několik typů zpracování obrazu (např. 3D detekce) podobně jako řešení vyvinuté na univerzitě Middlebury. KITTI se snaží o aplikaci v reálném světě. Toto řešení je dostupné na adrese „*http://www.cvlibs.net/datasets/kitti/i*“. [2]

3 Použité technologie

Tato kapitola je věnována nejdůležitějším technologiím použitým při vypracování této práce.

3.1 Python

Jedná se o dynamicky interpretovaný programovací jazyk vytvořený v roce 1990 Guido von Rossumem. Je objektově orientovaný a často bývá řazen mezi skriptovací programovací jazyky. Běží na většině platforem jako jsou UNIX, Windows, Mac OS. Python umožňuje tvorbu rozsáhlých plnohodnotných aplikací, včetně grafického rozhraní. Python je víceparadigmatický neboli hybridní jazyk. To sebou přináší možnost psát programy nejen v objektově orientovaném paradigma, ale i v procedurálním a v omezené míře i funkcionálním. Dále je unikátní svou čistou a jednoduchou syntaxí. Existuje ve dvou verzích 2.x a 3.x, které se liší syntaxí a jsou nekompatibilní. Pro vypracování této práce byla využita verze 2.7. [3]

3.2 Django Framework

Django je webový framework napsaný v Pythonu, který byl původně navržen pro správu několika webových stránek zpravodajské firmy The World Company. Vyvíjen je od roku 2003, ale až červnu 2005 byl veřejně vydán pod open-source licencí BSD. V dnešní době Django používají tisíce aplikací. Django vychází z architektonického vzoru MVC. Tento vzor interpretuje po svém a výsledkem je MTV.

Dále se snaží využívat koncept DRY, čímž se zamezuje rutinnímu opakování kódu. Využívá se objektově orientované programování a automatické generování kódu. Další silnou součástí Djanga je ORM přístup. To umožňuje pracovat pouze s objekty a funkcemi Pythonu. Toto ORM převádí objekty na dotazy a snaží se o jejich optimalizaci. Samozřejmostí je i možnost psát čisté SQL, což je výhodné pro složité dotazy. Využívá se implementace návrhového vzoru Active Record, kde každý model je potomkem třídy Model z modulu „django.db“. Django přináší také automaticky generovanou administraci. Díky této funkci je uživatelům umožněno jednoduše přistupovat a editovat data v databázi, bez nutnosti programovat vlastni administrační rozhraní. V neposlední řade je možnost spustit odlehčený vývojový webový server, na němž byla celá tato práce vyvíjena a testována.

Mezi další vlastnosti a funkce Djanga, které v této práci nebyly využity, patří:

- výkonný šablonovací systém,
- podpora kešování,
- generování formulářů z databázového modelu, automatická validace,
- škálovatelná architektura,
- lokalizace pomocí unixového nástroje gettext. [4], [5], [6]

3.2.1 Návrhový vzor Active Record

Jedná se o návrhový vzor pro práci s datovými zdroji, který uveřejnil Marin Fowler v roce 2002 ve své knize Patterns of Enterprise Application Architecture. Jde o objekt, který obaluje řádek v databázové tabulce, zapouzdřuje přístup k databázi a přidává doménovou logiku. Objekt nese data i chování. Většina těchto dat je trvalá a je potřeba je uložit v databázi. Active Record užívá nejjasnější přístup, vkládá logiku datového přístupu přímo do doménového objektu. Tímto způsobem je pro okolí zřejmé, jak číst a zapisovat data z a do databáze.

Podstatou Active Record je doménový model ve kterém se třídy shodují se záznamovou strukturou databáze. Jednotlivé třídy a jejich atributy reprezentují databázové tabulky a jejich atributy. Objekty představují jednotlivé řádky databázové tabulky. Každý Active Record je zodpovědný za ukládání a načítání dat z databáze (obstarává základní CRUD operace) a také za každou doménovou logiku, která působí na data. Toto může být všechna doménová logika aplikace, nebo také může být některá doménová logika držena v Transaction Script s běžným a objektově orientovaným kódem v Active Record.

Třída implementující Active Record má typicky metody, které obstarávají následující:

- vytváří instanci Active Record z výsledku SQL dotazu,
- vytváří novou instanci pro pozdější vkládání do tabulky,
- statické vyhledávací metody pro obalení běžně používaných SQL dotazů, které vracejí Active Record objekt,
- aktualizace a vkládání dat do databáze,
- získávání a nastavování jednotlivých atributů,
- implementace některé části bussines logiky [11]. [12]

3.3 FFmpeg

FFmpeg je multimediální framework, který umožňuje pokročilou práci s multimédii, jako je kódování, dekódování, filtrování atp. Výhodou je volná licence (LGP nebo GPL) a široká podpora mnoha formátů. FFmpeg je vyvíjen pod platformou GNU/Linux, ale může být zkompilován pod většinou operačních systému včetně Windows a Mac OS. Zakladatel projektu je Fabrice Bellard, avšak v současné době je udržován vývojářem jménem Michael Niedermayer.

FFmpeg nabízí nástroje:

- ffmpeg je nástroj příkazové řádky pro převádění multimediálních souborů mezi formáty,
- ffserver je multimediální streamovací server pro živé vysílání,
- ffplay je jednoduchý přehrávač médií založený na SDL a FFmpeg knihovnách,

- ffprobe multimediální stream analyzér. [7]

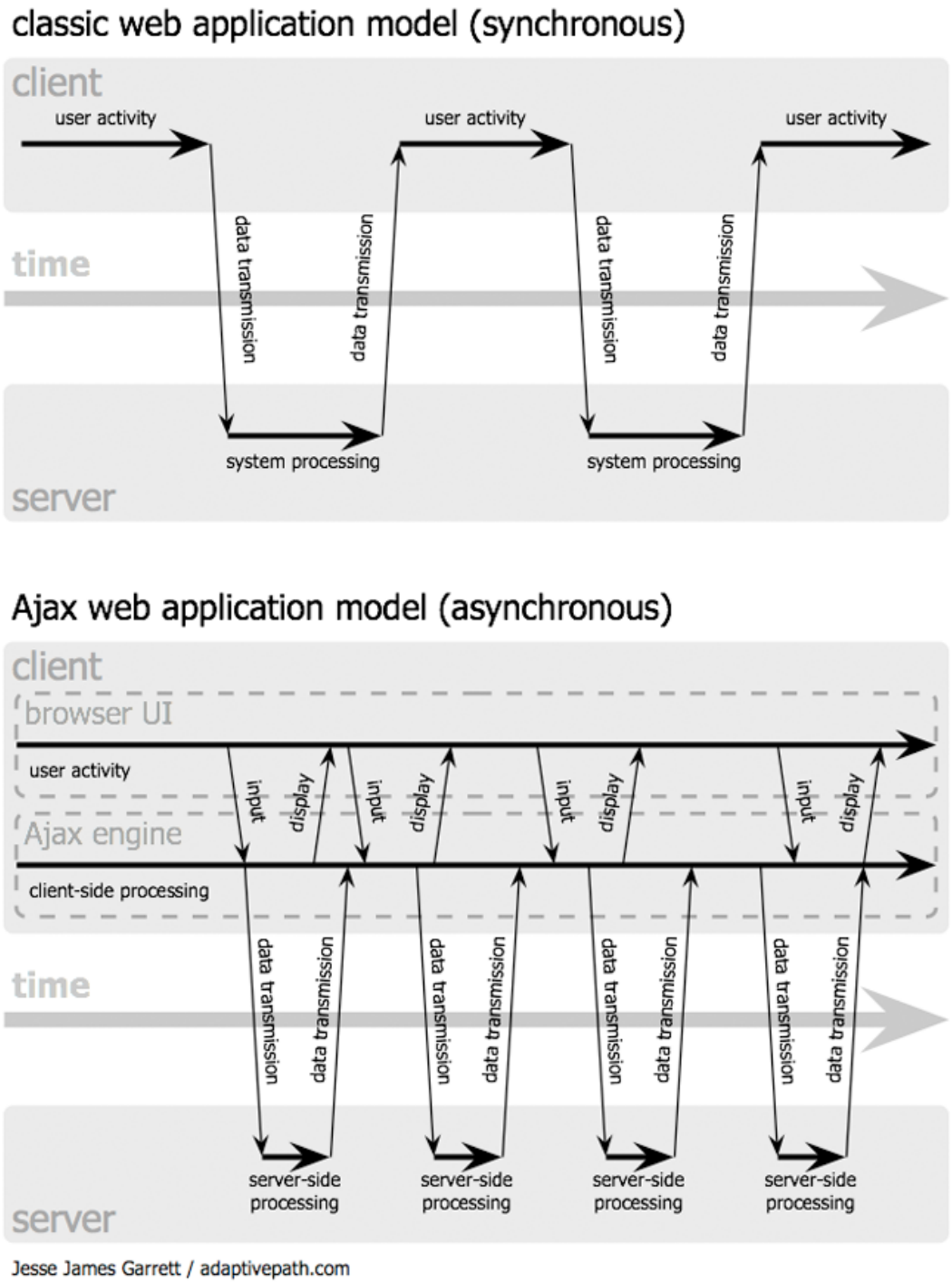
V této práci byl využit nástroj ffmpeg pro extrahování jednotlivých snímků z videa. Tato problematika je dále řešena v kapitole 4.

3.4 JavaScript

JavaScript je objektově orientovaný skriptovací jazyk, jehož autorem je Brendan Eich. Poprvé se objevil v roce 1995. Jeho hlavním smyslem byla validace vstupu na straně klienta, kterou do té doby obstarávala výhradně serverová strana jazyky jako je Perl. Jeho syntaxe patří do rodiny jazyku C/C++/C#/Java. JavaScript se obvykle provádí na straně klienta až po načtení stránky ze serveru. Z toho vyplývají některé bezpečnostní opatření, JavaScript nemůže například přistupovat k souborům na disku, aby neohrozil bezpečí uživatele. V současnosti tvoří základ dynamického webu. [10]

3.5 Asynchronous JavaScript and XML

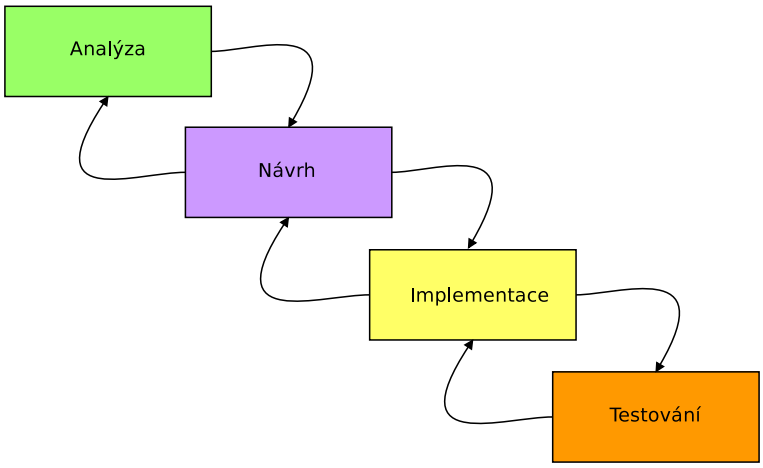
Asynchronous JavaScript and XML neboli AJAX není samostatný programovací jazyk, ale jde o způsob využití stávajících standardů. AJAX umožňuje výměnu dat se serverem a následnou aktualizaci části webové stránky bez nutnosti překladu celé stránky. Umožňuje zpracovat události asynchronně, což znamená, že události se zpracovávají na pozadí a neovlivňují hlavní tok aplikace. Aplikace, které nevyužívají AJAX, mohou vytvářet HTTP požadavky, ale po přijetí dat, musí být stránka znovu načtena a navíc v době mezi vygenerováním HTTP požadavku a přijetím dat ze serveru není možné vytvářet další požadavky. Oproti tomu, webové aplikace využívající AJAX mohou požadavky generovat neustále, příkladem takovýchto webových aplikací může být například „google maps“ nebo „facebook“. Porovnání mezi klasickou webovou aplikací a webovou aplikací využívající AJAX je graficky znázorněné na obrázku 1. [9]



Obrázek 1: Porovnání AJAX aplikace a klasické webové aplikace [9]

4 Samotné řešení

Tato kapitola je věnována vývoji práce od analýzy požadavků až po testování. Jako vzor pro postup řešení stál vodopádový model (se zpětnou vazbou) pro návrh software. Tento model nahlíží na vývoj softwaru jako na neustále svažující se tok fázemi v našem případě analýzy požadavků, návrhu, implementace a testování (viz obrázek 2). Díky zpětné vazbě je možnost vrátit se na předchozí fáze.



Obrázek 2: Vodopádový model vývoje software

4.1 Analýza

V této části budou zodpovězeny základní otázky pokládané při návrhu softwaru a dále specifikovány požadavky na systém.

4.1.1 Vize

Co? Jedná se o aplikaci, která by měla sloužit při vývoji algoritmů pro digitální zpracování obrazu. Měla by umožňovat otestovat tyto algoritmy a vzájemně porovnat jejich účinnost.

Pro koho je aplikace určena? Aplikace je cílena vývojářům algoritmů pro digitální zpracování obrazu.

Proč? Existuje mnoho algoritmů pro zpracovávání obrazu, nikoliv však aplikací, které by umožňovaly tyto algoritmy otestovat a vzájemně porovnat jejich účinnost. Z tohoto důvodu bude aplikace vyvíjena.

4.1.2 Specifikace požadavků

Existuje mnoho způsobů kategorizace specifikace požadavků. V tomto případě byla specifikace požadavků rozdělena na funkční a nefunkční požadavky.

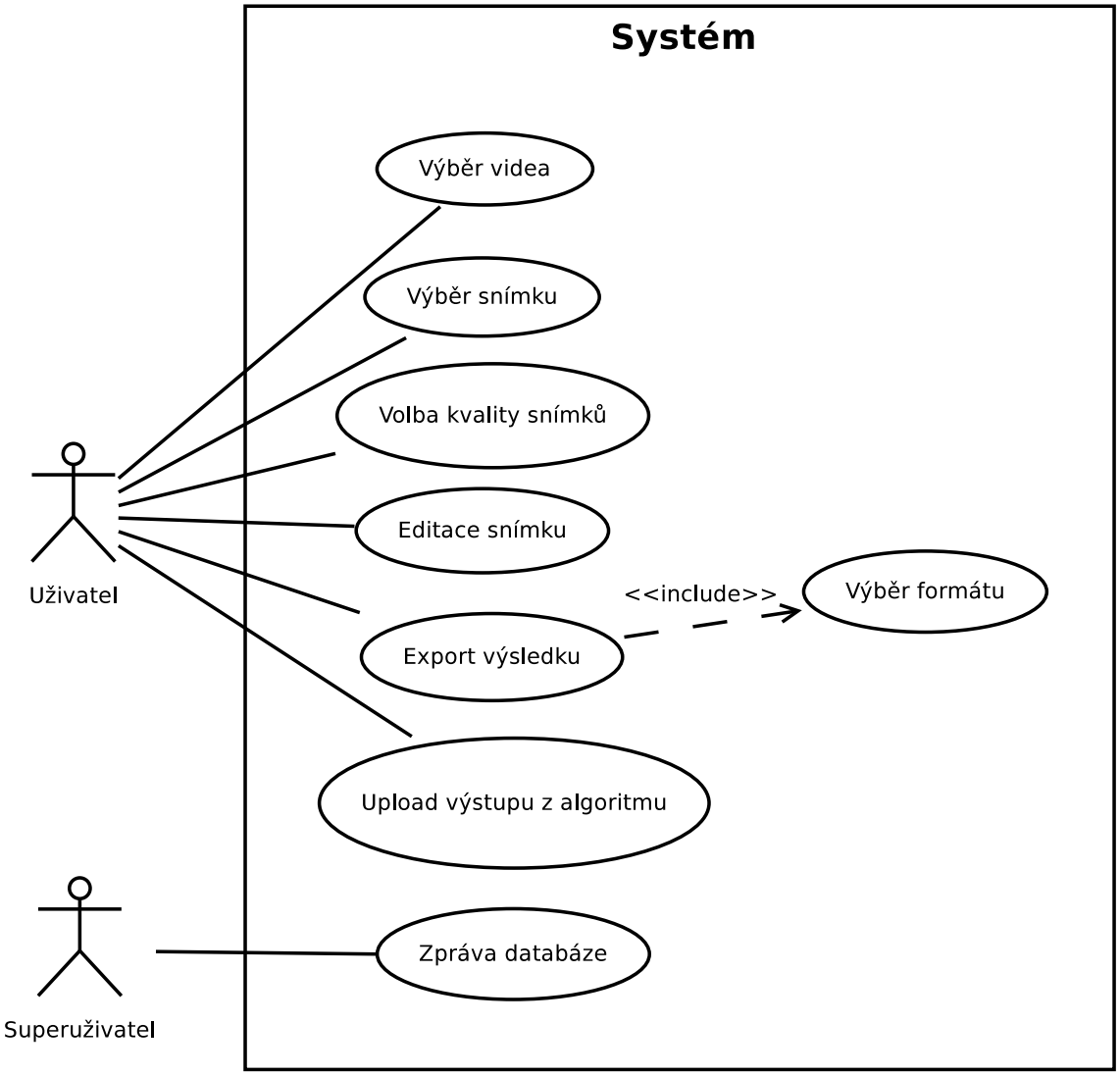
4.1.2.1 Funkční požadavky Funkční požadavky shrnují jednotlivé funkce systému a jsou uvedeny pomocí odrážek v následujícím seznamu.

- Výběr videa
  - Možnost zobrazit dostupné videa
- Výběr snímku
  - Uživatel bude moci vybrat libovolný snímek z videa
- Editace obsazenosti
  - Uživatel bude moci rozhodnout, zda je místo obsazené či nikoli
- Export výsledku
  - Možnost volby formátu exportovaného souboru:
    * CSV
    * XML
- Nahrání výstupu externích algoritmů
  - Podporované formáty:
    * CSV
    * XML
- Přehledný výpis srovnání externích algoritmů s ručně zadanými hodnotami
- Superuživatel bude moci spravovat databázi

4.1.2.2 Nefunkční požadavky Nefunkční požadavky představují podmínky uvalené na danou aplikaci.

- Využití Django Frameworku
  - databáze v SQLite
- Server na platformě Linux

Funkční strukturu systému graficky znázorňuje Use Case diagram (viz obrázek 3 na straně 13), jedná se o modelové znázornění.



Obrázek 3: Use Case diagram

4.2 Návrh

Tato část je věnována hlavně návrhu databáze potřebné pro ukládání dat poskytovaných prostřednictvím aplikace pro editaci obsazenosti parkoviště a dat získaných porovnáním těchto ručně zadaných hodnot s hodnotami získanými ze souborů nahraných prostřednictvím aplikace pro příjem dat z algoritmů zpracování obrazu. Tato část se věnuje také návrhu uživatelského rozhraní pro obě zmíněné aplikace.

4.2.1 Databáze

Pro databázi byly navrhnuty 2 tabulky. Tabulka „Park“ je určena pro ukládání stavu parkoviště pro jednotlivé snímky a videa. Ve druhé tabulce „Dataset“ se budou ukládat výsledky porovnání dat z algoritmů pro digitální zpracování obrazu a ručně zadané hodnoty prostřednictvím aplikace pro editaci obsazenosti parkoviště.

Tyto tabulky nemají žádný vzájemný vztah, aby byly vzájemně nezávislé. Obě tyto tabulky obsahují atribut „video“, který představuje název videa. Tato vzniklá redundance je nutná, aby jsme měli k dispozici v obou tabulkách název videa. Této redundanci by bylo možné předejít vytvořením vzájemného vztahu mezi tabulkami nebo přidáním třetí tabulky a k ní příslušných vztahů, která by obsahovala informace o jednotlivých videích. Tímto by jsme ale přišli o nezávislost mezi tabulkami, čímž by jsme při vymazáni kořenového záznamu nesoucího název videa ztratili informaci o názvu videa i v ostatních tabulkách. Takovéto záznamy by bylo dále zbytečné uchovávat. Jelikož ale v tabulce „dataset“ chceme uchovávat výsledky i u videí, které jsou již smazány, tak je tato možnost nepřístupná.

4.2.1.1 Datový slovník Podrobnější náhled na entity a jejich atributy je zobrazen v datovém slovníku. Datový slovník pro tabulku „Park“(tabulka 2) je uveden v příloze na straně 32. Tabulka 1 na straně 14 reprezentuje datový slovník pro entitu „Dataset“.

Dataset				
Název	**Typ**	**Délka**	**PK**	**Popis**
id	int	-	ano	primární klíč
video	char	100	ne	název videa
frames	int	-	ne	počet snímku
Algorithm	char	100	ne	název algoritmu
NumOfcorrDet	int	-	ne	počet správných detekcí
NumOfBadDet	int	-	ne	počet špatných detekcí

Tabulka 1: Datový slovník - Dataset

4.2.2 Návrh uživatelského rozhraní

Cílem bylo navrhnout jednoduché, intuitivní uživatelské rozhraní pro editování obsazenosti parkoviště a pro aplikaci pro příjem dat z algoritmů digitálního zpracování obrazu. Při návrhu uživatelského rozhraní se vychází z funkčních požadavků a je nutné, aby všechny tyto požadavky byly splněny.

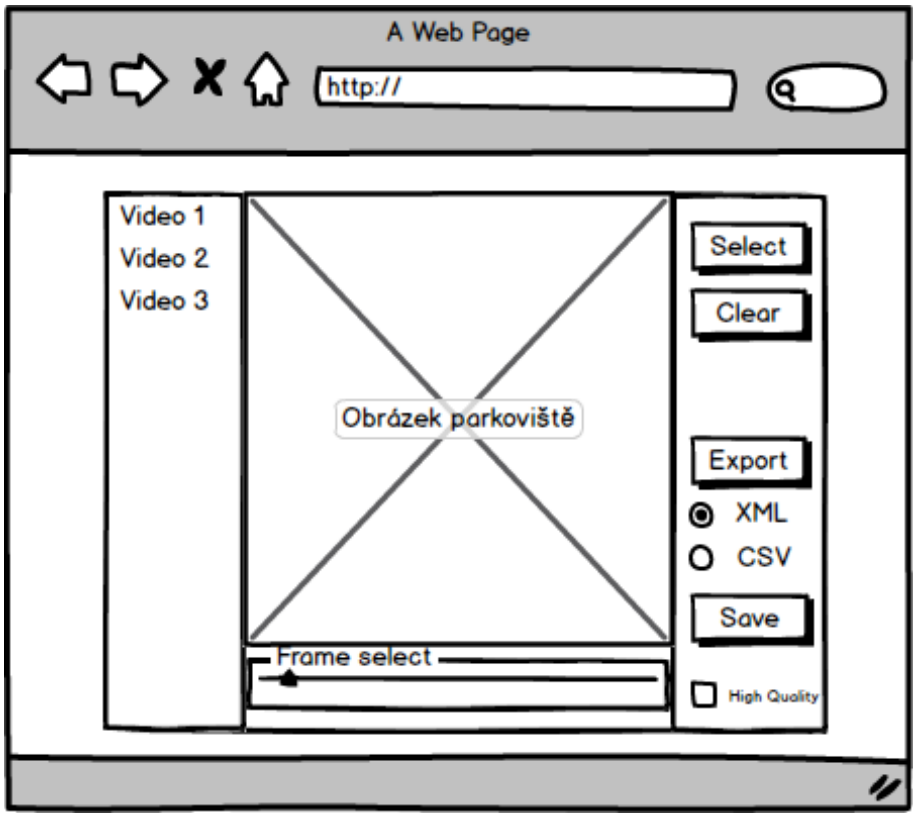
4.2.2.1 Aplikace pro editaci obsazenosti parkoviště Levá část aplikace bude sloužit pro výpis dostupných videí. V pravé části budou ovládací tlačítka umožnující ukládání, exportování výsledků a dále tlačítka pro označení a odznačení všech parkovacích míst.

V této části bude také „checkbox“ pro volbu kvality videa a seznam typu „radio button“ pro volbu formátu exportovaného výsledku. V dolní části aplikace byl navrhnut posuvník, kterým bude možno volit jednotlivé snímky videa. Ve střední části aplikace se budou zobrazovat jednotlivé snímky z videa. Výsledný návrh je zobrazen na obrázku 4.

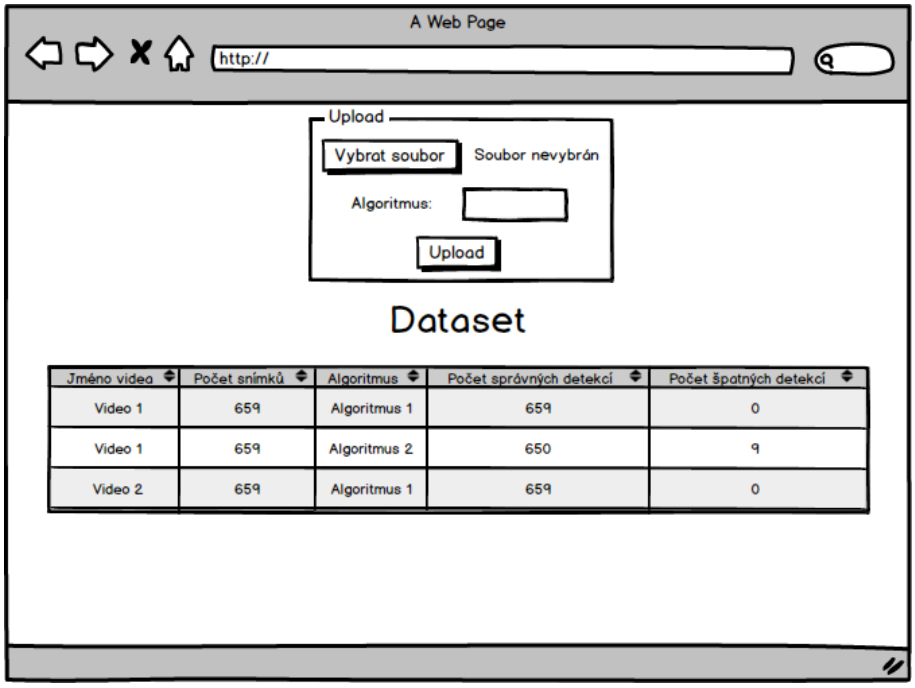
4.2.2.2 Aplikace pro příjem dat z algoritmu digitálního zpracování obrazu

Dále bylo nutné navrhnout uživatelské rozhraní k aplikaci umožňující nahrávání souboru s výstupy algoritmů pro digitální zpracování obrazu a zobrazování výsledků porovnání ručně zadaných hodnot pomocí aplikace pro editaci obsazenosti parkoviště s hodnotami získanými z nahraných souborů.

V horní části aplikace byl navrhnut formulář pro nahrávání souboru s daty z algoritmů pro digitální zpracování obrazu. Pro zobrazení výsledků byla navrhnuta tabulka, která se rozkládá na zbývajícím místě. V tabulce by měla být možnost setřídit data dle libovolného sloupce. Výsledný návrh je zobrazen v obrázku 5.



Obrázek 4: UI aplikace pro editaci obsazenosti parkoviště



Obrázek 5: UI aplikace pro příjem dat z algoritmů zpracování digitálního obrazu

4.3 Implementace

Tato sekce je logicky rozdělená na dvě části, a to na stranu serveru a stranu klienta. Bude se věnovat samotné implementaci a problémům, který bylo nutné řešit.

4.3.1 Strana serveru

Serverová strana je napsaná v jazyce Python za využití Django Frameworku. Stará se o obsluhu požadavků generovaných na straně klienta a o práci s databázi. Pro databázi je využito SQLite3. Databáze využívající SQLite3 je umístěna v jednom souboru nezávislém na platformě, což přináší výhody jako jsou jednoduchost a přenositelnost. Díky ORM přístupu Django Frameworku je databáze generována z jednotlivých modelů, které reprezentují třídy dědící z „django.db.Models.model“, které jsou umístěny v souboru „models.py“. Třídní diagram reprezentující databázové schéma je umístěn v příloze na straně 34 pod číslem 12.

4.3.1.1 Zpracování požadavků Pro zachytávání požadavků jsou v Djangu tzv. wiev funkce. Tyto funkce jsou namapované na konkrétní URL. Když přijde požadavek na konkretní URL (například „http://127.0.0.1:8000/getImg“), tak Django načte soubor s konfiguraci URL vzorů. Následně projde každý URL vzor v tomto souboru a porovnává ho s URL požadavku dokud nenajde shodu. Když najde shodu, tak zavolá view funkci přidruženou k tomuto URL a předá ji objekt typu HttpRequest jako první parametr.

V tomto objektu jsou nám zpřístupněna metadata o HTTP požadavku prostřednictvím jeho atributů a metod.

Odpověď je reprezentována instancí třídy HttpResponse. U tohoto objektu máme možnost vyplnit všechny odpovědní hlavičky HTTP protokolu.[5]

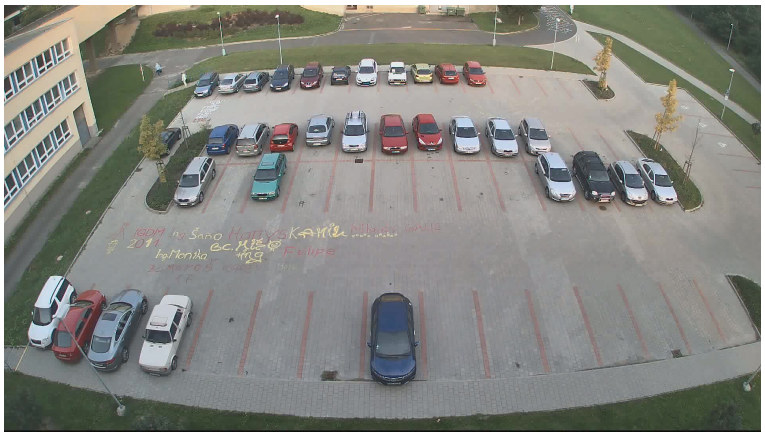
Příkladem může být funkce „getVideos“ (viz výpis 1). Tato funkce vrací seznam videí uložených na serveru.

```
def getVideos(request):
    videos = filter (lambda x: x.endswith(".mp4"), os. listdir (" ./ media"))

    jvideos = json.dumps(videos)

    response = HttpResponse(jvideos)
    response["Access-Control-Allow-Origin"] = "*"
    return response
```

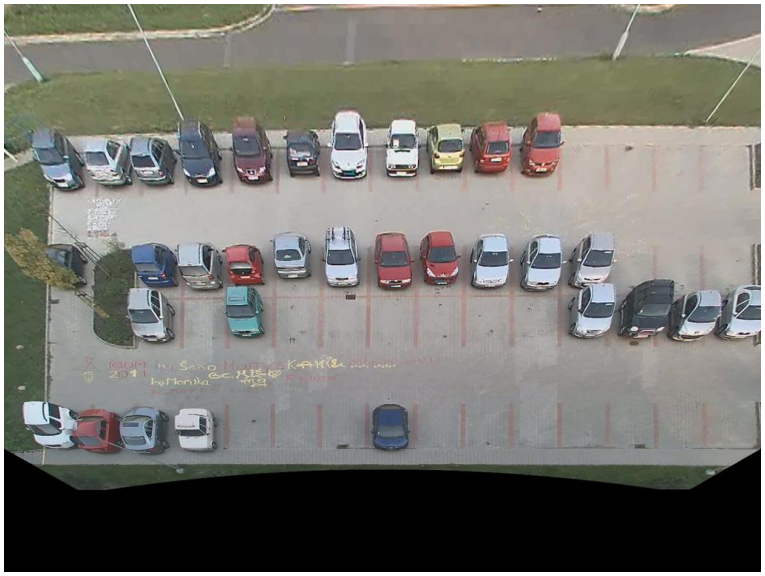
Výpis 1: Funkce getVideos

4.3.1.2 Extrahování snímku z videa Jelikož jsou na serveru uložená celá videa, je nutné z nich extrahovat jednotlivé snímky. K tomuto účelu je využit FFmpeg, konkrétně jeho nástroj pro příkazovou řádku, ffmpeg. Pro spouštění příkazu v příkazové řádce je využit Python modul „subprocess“. Tento modul obsahuje metodu „call“, která přebírá příkaz pro příkazovou řádku jako parametr. Použití je možné vidět ve funkci „getImg“ ve výpisu 2.

Extrahovaný snímek, který vidíme na obrázku 6, je nutné upravit pomoci programu „Korekce perspektivní projekce“, jehož autorem je Ing. Jan Gaura. Tento program odstraní zkreslení způsobené objektivem a snímek perspektivně narovná. Funkce tohoto programu bude detailněji vysvětlená v odstavci 4.3.1.3. Výsledný snímek (obrázek 7) je již možno použít pro zobrazení v klientské aplikaci.



Obrázek 6: Neupraveny snímek



Obrázek 7: Upravený snímek

Pro extrahování snímku z videa slouží funkce „getImg“ (výpis 2 na straně 18). Tato funkce vrací konkrétní snímek z videa.

```
def getImg(request):
    i = int(request.GET.getlist("scr")[0])
    video = request.GET.getlist("vid")[0]
    quality = request.GET.getlist("quality")[0]

    fps = getFps(video)
    time = float(i)/fps
    if quality == "low":
        cs = "ffmpeg -vframes 1 -y -ss "+str(time)+" -i ./media/"+video+" -s 1366x768 ./
            media/output.jpg"
    else:
        cs = "ffmpeg -vframes 1 -y -ss "+str(time)+" -i ./media/"+video+" ./media/output.jpg"
    subprocess.call(cs, shell=True)

    cs = "cd media/narovnavac; ./narovnavac "+quality+";"
    subprocess.call(cs, shell=True)

    size = 1024, 768
    im = Image.open("./media/narovnavac/strecha1_persp.jpg")
    im_resized = im.resize(size, Image.ANTIALIAS)
    im_resized.save("./media/narovnavac/my_image_resized.jpg", "JPEG")

    data_uri = open("./media/narovnavac/my_image_resized.jpg", "rb").read().encode("base64").
        replace("\n", "")
    img_tag = "{0}".format(data_uri)
```

```
response = HttpResponse(img_tag)
response["Access−Control−Allow−Origin"] = "∗"
return response
```

Výpis 2: Funkce getImg

4.3.1.3 Korekce perspektivní projekce Jedná se o program vyvinutý v jazyku C++, který odebírá zkreslení objektivu (tzv. distorze objektivu) a následně narovnává snímek. Autorem tohoto programu je Ing. Jan Gaura.

Vlivem optických parametrů objektivu mohou vznikat geometrické distorze obrazu způsobené jeho optikou. Existuje několik typů geometrické distorze obrazu. V tomto případě se jedná o typ barel. Tuto distorzi způsobuje vlastnost širokoúhlých objektivů, díky které je střed obrazu více přiblížen než okolí.

Distorze typu barel patří mezi tzv. radiální zkreslení. Jeden ze způsobů jak toto zkreslení odebrat, je pomocí posunu souřadnic obrazu. Pro bod ve zkresleném obrazu mějme souřadnice $q = (x_d, y_d)^T$ a pro bod v nezkresleném obrazu mějme souřadnice $\overline{q} = (x_n, y_n)^T$. Pomocí Taylorova polynomu lze aproximovat každé radiálně symetrické zkreslení.

$$\emptyset(r^2) = 1 + K1r^2 + K2r^4 + \ldots,$$

kde $r^2 = \overline{x}^2 + \overline{y}^2$ a $K1, K2 \ldots$ jsou koeficienty radiálního zkreslení. Díky bezrozměrnosti souřadnic $\overline{x}$, $\overline{y}$ není podstatná velikost vstupního obrazu. Je také nutné přesunout počátek souřadnicového systému do středu obrazu, kvůli zachování radiální symetrie. Tyto souřadnice je možné získat z následujících rovnic:

$$\overline{x} = \frac{x_n - c_u}{R}$$

$$\overline{y} = \frac{y_n - c_v}{R},$$

kde $R = \sqrt{c_u^2 + c_v^2}$ a souřadnice středu obrazu $c_u = w/2$, $c_v = h/2$ (w. . . šířka obrazu, h. . . výška obrazu).

Následující rovnice popisuje přechod ze souřadnic x_n, y_n obrazu nezkresleného do souřadnic x_d, y_d obrazu zkresleného.

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \begin{pmatrix} x_n - c_u \\ y_n - c_v \end{pmatrix} \emptyset^{-1}(r^2) + \begin{pmatrix} v_u \\ c_v \end{pmatrix}$$

Dále je nutné procházet snímek po jednotlivých pixelech a vypočítávat správné souřadnice jednotlivých pixelů. Na výsledné souřadnice je nutné použít vhodnou interpolaci. [13]

Program korekce perspektivní projekce využívá výše popsanou metodu pro odstranění distorze obrazu. Snímek po odstranění distorze můžeme vidět v příloze na straně 33 pod číslem 10. Na takto upraveném snímku je dále provedena operace „narovnání

perspektivy“, k tomu jsou využity funkce „cvGetPerspectiveTransformation“ a „cvWarpPerspective“ z knihovny pro práci s grafikou „opencv“.

Funkce „cvGetPerspectiveTransformation“ vypočítá perspektivní transformaci ze čtyř odpovídajících bodů. Tato funkce má tři parametry, parametr „src“, který reprezentuje pole souřadnice čtyř vrcholů čtyřúhelníku zdrojového obrazu, dále parametr „dst“, který představuje souřadnice ze čtyř odpovídajících vrcholů čtyřúhelníku cílového obrazu a parametr „matMatrix“, ukazatel na cílovou 3x3 matici. Funkce počítá matici perspektivní transformace tak, že:

$$\begin{bmatrix} x'_i \\ y'_i \end{bmatrix} = matMatrix \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix},$$

kde

$$dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0, 1, 2, 3$$

Funkce „cvWarpPerspective“ aplikuje perspektivní transformaci na obraz. Přijímá čtyři parametry a to parametr „src“, který představuje zdrojový obraz, dále parametr „dst“ který reprezentuje cílový obraz, parametr „mapMatrix“, který představuje 3x3 transformační matici a parametr „flags“, který představuje interpolační metodu. [14]

Narovnaný snímek se nachází v příloze na straně 33 pod číslem 11. Funkce pro odstranění distorze je zobrazena ve výpisu 3.

```
void RemoveLensDistortion()
{
    IplImage * dst32FC3 = cvCreateImage( cvSize( image->width, image->height ),
            IPL_DEPTH_32F, 3 );

    const TYPE_REAL K1 = 0.534;
    const TYPE_REAL K2 = -0.070;
    const TYPE_REAL zoom = 1.398;
    const double cu = image->width / 2.0;
    const double cv = image->height / 2.0;
    const double R = sqrt( cu*cu + cv*cv );

    for ( int x = 0; x < image->width; x++ )
    {
        for ( int y = 0; y < image->height; y++ )
        {
            double u = ( x - cu ) / R;
            double v = ( y - cv ) / R;
            double r2 = u*u + v*v;
            double rdc = ( 1.0 + K1 * r2 + K2 * r2*r2 );

            u = zoom * ( x - cu ) / rdc + cu;
            v = zoom * ( y - cv ) / rdc + cv;

            CvScalar pixel = BilinearInterpolation3U ( image, u, v );

            float * p = &CV_IMAGE_ELEM( dst32FC3, float, y, x*image->nChannels );
```

```
                p[0] = pixel.val[0] / 255;
                p[1] = pixel.val[1] / 255;
                p[2] = pixel.val[2] / 255;

            }
        }

        img=dst32FC3;
    }
```

Výpis 3: Funkce pro odstranění zkreslení

4.3.1.4 Exportování výsledku Aplikace umožňuje uživatelům stahovat zadané výsledky ve dvou různých formátech. Prvním podporovaným formátem je XML.V tomto formátu jsou data strukturována do stromové struktury. Struktura XML souboru je zobrazena ve výpisu 4. Skládá se z uzlů:

- parking - kořenovy uzel,
- video - reprezentuje celé video v databázi, má atribut „name“,
- frame - představuje jeden snímek videa, má atribut „number“,
- p - představuje jedno místo parkoviště, má atributy „id“ a „value“.

Pro práci s XML soubory je využita knihovna „ElementTree“, která obsahuje potřebné funkce pro vytváření a čteni XML souboru.

```
<parking>
    <video name="2012-09-05_08.mp4">
        <frame number="1">
            <p id="1" value="False"/>
            <p id="2" value="False"/>
            .
            .
            .
        </frame>
    </video>
</parking>
```

Výpis 4: Struktura exportovaného xml souboru

Druhým podporovaným formátem je CSV. V tomto formátu jsou data strukturována do řádku a oddělena oddělovačem. Jako oddělovač zde slouží „ , “ a jeho konkrétní struktura je: název videa, číslo snímků a dále stav jednotlivých parkovacích míst zapsaných booleovými hodnotami (True - volno, False - obsazeno). Pro práci s CSV formátem je využit Python modul „csv“.

Při požadavku na export výsledku je zavolána funkce „export“, která přebírá parametr typu HttpRequest. Z tohoto parametru je získán požadovaný typ exportovaného souboru.

Dále jsou z databáze načtená všechna data z tabulky „Park“. V případě, že se exportuje ve formátu CSV, jsou tato data zapisována po řádcích podle struktury, která je zmíněna výše. V případě, že se exportuje do XML souboru, jsou vytvářeny uzly s daty podle struktury popsané výše.

4.3.1.5 Zpracování nahraných výstupu z externích algoritmů

Při nahrání souboru na server je volána metoda „upload“, která jako parametr přijímá objekt typu HttpRequest. V tomto objektu je obsažen nahraný soubor a název algoritmu. Dále je rozhodnuto pomocí koncovky souboru, zda-li se jedná o XML nebo CSV soubor.

Pokud se jedná o CSV soubor, kde jsou data k jednotlivým snímkům uložena v řádcích, je tento soubor rozparsován a po jednotlivých řádcích se porovnávají hodnoty z tohoto souboru s hodnotami v databázi pomocí metody „compare“. V případě, že se jedná o XML soubor, je potřeba soubor opět rozparsovat. Data k jednotlivým snímkům jsou získávána pomocí dvou vnořených cyklů, kde vnější cyklus prochází jednotlivé uzly „video“ a vnořený cyklus prochází jednotlivé uzly „frame“. Takto získaná data jsou porovnávána pomocí funkce „compare“, stejně jako v případě, kdy se jedná o CSV soubor.

Funkce „compare“ přijímá tři parametry (vid - název videa, frame - číslo snímků, pole - pole reprezentující stav parkoviště pro daný snímek). Z databáze si načte data pro příslušné video a snímek. Tato data dále porovnává s daty, které ji byly předány pomocí parametru. Při schodě dat je inkrementována proměnná reprezentující počet správných detekcí a v případě, že jsou data rozdílná, je inkrementovaná proměnná reprezentující počet špatných detekcí. Takto získáme počet správných a špatných detekcí pro určitý snímek. Tyto hodnoty jsou vráceny jako pole zpět do funkce „upadate“, kde jsou přičteny k celkovému počtu správných a špatných detekcí.

Po průchodu všech snímků videa je uložen výsledek v databázi v tabulce „Dataset“. V případě, že jsou data nekompletní, ať už v nahraném souboru nebo v databázi, výsledek porovnání se neukládá.

```
def compare(vid, frame, pole):

    p = Park.objects.filter(video = vid, screen = frame)

    s = p.values_list()
    row = s[0]
    detection = [0,0]

    for i in range(len(row[3:])):

        if str(pole[i]) == str(row[i+3]):
            detection[0]+=1
        else:
            detection[1]+=1

    return detection
```

Výpis 5: Funkce compare

4.3.1.6 Podrobný popis funkcí

V této části budou vyjmenovány a detailně popsány veškeré naimplementované funkce, které dosud nebyly zmíněny.

Funkce getImg Tato funkce při zavolání dostává jako parametr objekt typu HttpRequest. Z tohoto objektu si získá atributy číslo snímku, název videa, kvalita videa. Dále je nutno zjistit, v jakém čase videa se požadovaný snímek nachází. Toto se provede jednoduchým výpočtem:

$$t = \frac{fps}{i},$$

kde t představuje čas snímku, fps představuje framerate videa a i je číslo požadovaného snímku. Hodnotu framerate z videa získáme pomoci funkce „getFps“, která zjistí framerate videa a vrátí ho jako návratovou hodnotu.

Podle atributu představujícího kvalitu videa je rozhodováno o rozlišení, v jakém bude snímek extrahován. V případě nízké kvality je snímek extrahován v rozlišení 1366x768 pixelů a v případě kvality vysoké se snímek extrahuje v rozlišení 1920x1080 pixelů. Nyní už máme všechny potřebné informace pro extrahování snímku z videa, takže je sestaven ffmpeg příkaz a pomocí metody „call“ z modulu „subprocess“ zavolán.

Extrahovaný snímek je dále nutné upravit programem „korekce perspektivní projekce“. Funkce tohoto programu je detailněji rozepsána v následující sekci. Tento program je spuštěn z příkazové řádky také pomocí metody „call“ z modulu „subprocess“ a je mu předán parametr kvality. Takto získáme snímek požadované kvality, který je pouze nutné upravit na rozlišení vhodné pro aplikaci pro editaci obsazenosti parkoviště.

Jako vhodné rozlišení bylo zvoleno 1024x768 pixelů. Tato změna je provedena pomocí metody „resize“ z Python knihovny PIL. Takto upravený snímek je převeden do formátu base64 a pomocí objektu HttpResponse vrácen klientské aplikaci.

Funkce getNumOfFrames Úkolem této funkce je zjistit počet snímků ve videu a vrátit tento údaj jako návratovou hodnotu. Tato funkce má jeden parametr a to video, který reprezentuje název videa, ze kterého chceme zjistit počet snímků. Pomocí metody „popen“ z modulu „subprocess“ je zavolán ffmpeg s přepínačem „-i“, parametrem název videa a s linuxovými příkazy, které z výpisu informací o videu vytáhnou pouze délku videa. Tento získaný údaj je ve formátu „HH:MM:SS“. Počet snímku videa získáme pomocí vypočtu:

$$frames = ((h \cdot 60 + m) \cdot 60 + s) \cdot fps,$$

kde h představuje hodiny, m minuty, s sekundy, fps framerate videa. Framerate videa je získán pomocí funkce getFps.

Funkce getFps Tato funkce je používána pro zjištění framerate daného videa. Přijímá parametr video, který představuje název videa, ze kterého je zjišťována hodnota framerate. Pomocí metody „popen“ z modulu „subprocess“ je zavolán ffmpeg s přepínačem „-i“, parametrem název videa a s linuxovými příkazy, které z výpisu informací o videu vytáhnou hodnotu framerate. Tento údaj je převeden na datový typ float a vrácen jako návratová hodnota funkce.

Funkce getVideos Funkce je volána při obsluze požadavku na seznam dostupných videí. Pro získání těchto údajů se využívají Python funkce „lambda“ a „filter“. Tímto způsobem dostaneme pole souboru v definované složce typu mp4. Toto pole je převedeno do formátu JSON a vráceno jako objekt typu HttpResponse.

Funkce getDataset Tato funkce na požadavek o dataset vrátí XML data formátovaná ve tvaru HTML tabulky. Pomocí Python knihovny „ElementTree“ je vytvořena hlavička tabulky. Dále jsou načtena data z databáze a v cyklu ukládána do XML uzlů reprezentujících řádky tabulky. Takto vytvořená XML data jsou vrácena pomocí objektu typu HttpResponse.

Funkce save Funkce save slouží pro obsloužení požadavků na uložení dat z aplikace pro editaci obsazenosti parkoviště do databáze. Této funkci je předán parametr typu HttpRequest, ze kterého si tato funkce vytáhne data potřebná pro uložení do databáze a předá je funkci insert.

Funkce load Tato funkce vrací stav parkoviště pro snímek, který je zrovna zobrazován v aplikaci pro editaci obsazenosti parkoviště. Přijímá parametr typu HttpRequest, ze kterého si získá název videa a číslo snímku. Pokud záznam s takovým názvem videa a číslem snímku existuje, je načten z databáze a vrácen pomocí objektu HttpResponse.

Funkce insert Tato funkce se stará o vkládání dat do databáze. Přijímá čtyři parametry (x - číslo snímku, na kterém byla naposledy provedena změna stavu parkoviště, y - číslo snímku současného neboli snímku, ze kterého byl požadavek na uložení do databáze generován, pole - pole reprezentující stav parkoviště, vid - parametr reprezentující název videa). Při ukládání jsou v cyklu v rozsahu x až y vkládány data do databáze. Pokud záznam v databázi již existuje, je aktualizován.

Funkce insertIntoDataset Tato funkce přijímá čtyři parametry (vid - parametr reprezentující název videa, alg - parametr představující název algoritmu a parametry noc, nob, které představují počet špatných a počet dobrých detekcí), které uloží do databázové tabulky „Dataset“.

4.3.2 Strana klienta

Klientská strana aplikace se skládá ze dvou částí. První reprezentuje aplikaci pro editaci obsazenosti parkoviště a druhá reprezentuje aplikaci pro příjem dat z algoritmů zpracování obrazu. Aplikace jsou realizovány pomocí webových stránek strukturovaných pomocí HTML kódu. O grafickou stránku aplikace se stará externí CSS soubor. Aplikace reagují na uživatelem vyvolané události, pomocí JavaScriptových funkcí. Příkladem takové události může být událost „onclick“. Jak už název napovídá, tak tato událost se vyvolá po kliknutí na objekt, ke kterému je přiřazena.

V aplikacích je využito některých nových prvků specifikace HTML5. Jedním z těchto prvků je tag input typu range, který v aplikaci realizuje posuvník obrázků. Nevýhodou využívání nových prvků v HTML5 je, že ne všechny tyto prvky jsou podporovány všemi prohlížeči.

4.3.2.1 Komunikace se serverem Komunikaci se serverem zajišťují AJAX funkce. Konkrétně jsou to funkce „POST“ a „GET“. Pro zjednodušení zápisu funkcí je využitá JavaScriptová knihovna JQuery. Příkladem může být funkce „getImg“ (výpis 6), která zasílá žádost „GET“ na adresu „ *http://localhost:8000/getImg* “ s parametry: číslo snímku, název videa, kvalita. Server vrátí snímek ve formátu base64 (formát reprezentující binární data pomoci znaku ASCII), který je následně umístěn na stránku.

```
function getImg(val){
        document.getElementById("loader").style.display ="block";
        var qual = "";
        if (! $("#quality ") . is (': checked')){
            qual = "low";
        }
    $.get("http :// localhost:8000/getImg", {scr: val, vid: video, quality : qual }, function(data) {
            document.getElementById("screen").style.backgroundImage="url("+data+")";
            document.getElementById("loader").style.display ="none";
    });
}
```

Výpis 6: Funkce getImg

4.3.2.2 Aplikace pro editaci obsazenosti parkoviště Po spuštění aplikace je na událost „onload“volána funkce „initialize“. Tato funkce nastaví potřebná data a zavolá funkci „getVideos“, která posílá na server požadavek na seznam dostupných videí. Server vrátí pole videí, ze kterého se vytvoří seznam videí, kde jednotlivá videa reprezentují odkazy, které mají přidělenou anonymní funkci k události „onclick“, která provede volbu videa. Tento seznam je umístěn do levého sloupce aplikace.

Po zvolení videa je zavolána funkce „getSlider“, která volá požadavek o počet snímku ve videu na server. Server vrátí tuto hodnotu, díky které se nastaví krajní hodnoty posuvníku pro volbu snímku. Dále je volána funkce „getImg“(neplést s funkci „getImg“, která je implementovaná na straně serveru), která volá požadavek s parametry určující požadovaný snímek videa (v tomto případě jde o snímek první). Tento snímek je následně umístěn na pozadí příslušného prvku. Volit snímky videa je možné pomocí dříve zmíněného posuvníku. Tento posuvník vyvolává událost „onchange“ a událost „onmouseup“. Událost „onchange“ volá metodu „ShowValue“, která mění pouze čísla indikátoru snímku. Na událost „onmouseup“ je volána funkce „changeImg“. Tato funkce pouze zavolá výše zmíněnou metodu „getImg“ a následně metodu „load“, která, pokud byl již snímek zaznačen, získá ze serveru data o stavu parkoviště a zobrazí jej. Tento posuvník lze použít pro volbu snímku v obou směrech, díky tomuto existuje možnost kontroly dříve zaznačených snímků.

Parkoviště se skládá z 58. parkovacích míst, aby zaznačování plných míst bylo co nejvíce usnadněno, je naimplementována možnost zaznačovat místa hromadně pomocí funkce „StartDragSelect“. Tato funkce umožňuje zaznačovat místa pomocí přidržení stisknutého levého tlačítka myši a tahem na místa, která chceme zaznačit. Dále jsou naimplementována tlačítka, která umožňují zaznačit nebo odznačit všechna místa parkoviště najednou. K těmto tlačítkům jsou přiřazeny funkce „selectAll“ a „unselectAll“. Tyto funkce procházejí pole reprezentující stav parkoviště a nastaví všechny prvky pole na hodnotu „true“ v případě funkce „unselectAll“ a v případě funkce „selectAll“ jsou prvky tohoto pole nastaveny na hodnotu „false“.

Ukládání probíhá automaticky, vždy když se stav současného snímku liší od předchozího. Toto se testuje vždy při přechodu na další snímek, proto je nutné na posledním snímku práci uložit pomocí tlačítka „save“. Na server se pak posílají data ve formátu JSON a obsahují: číslo snímku u kterého nastala poslední změna, číslo současného snímku, stav parkoviště a název videa.

Další funkcí aplikace je možnost exportovat výsledné zaznačení z databáze. V pravém sloupci aplikace je pro tuto funkci naimplementováno tlačítko „export“ a seznam typu „radio button“ pro volbu formátu exportovaných dat. Po kliknutí na toto tlačítko je na událost „onclick“ zavolána funkce „download“. Tato funkce generuje požadavek exportování na server. Server pošle příslušná data, která jsou v případě XML formátu zobrazena v novém okně a v případě CSV formátu je otevřeno dialogové okno pro stažení souboru.

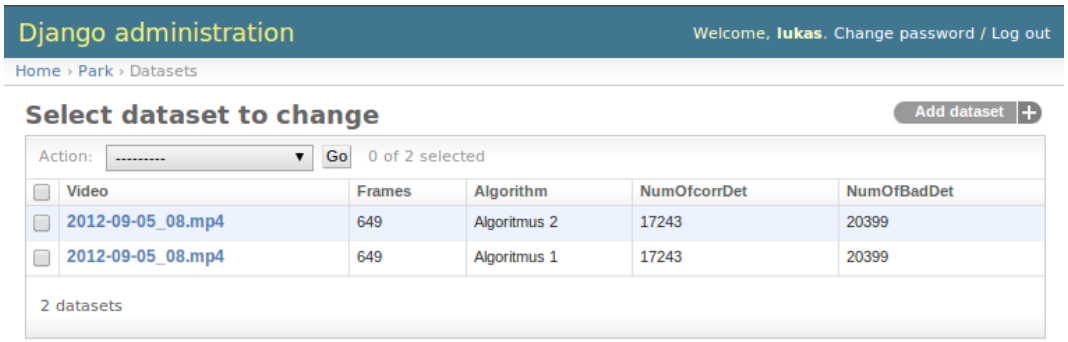
Obrázek 8: Aplikace pro editaci obsazenosti parkoviště

4.3.2.3 Aplikace pro příjem dat z algoritmů zpracování obrazu

Tato aplikace zobrazuje výsledky porovnání algoritmů zpracování obrazu s ručně zadanými hodnotami pomoci aplikace pro editaci obsazenosti parkoviště. A dále umožňuje nahrávat výsledky algoritmu pro digitální zpracování obrazu. Po spuštění aplikace je na událost „onload“ zavolaná funkce „getDataset“, tato funkce generuje požadavek na server a očekává data už strukturovaná pomocí HTML kódu do tabulky, kterou umístí na stránku. Díky JavaScriptové knihovně „sorttable.js“ je umožněno seřadit tabulku dle libovolného sloupce pouhým kliknutím na hlavičku tabulky. Pro nahrávání výsledků je v horní části aplikace umístěn formulář, který umožňuje zadat jméno algoritmu a přiložit soubor s výstupem z algoritmu pro digitální zpracování obrazu. Pro nahrávané soubory jsou podporované formáty XML a CSV. Struktura XML a CVS souboru viz odstavec „Exportování výsledku“. Samotné nahrání souboru je možné provést pomocí tlačítka „upload“, které vyvolává událost „onclick“, na kterou reaguje funkce „upload“. Tato funkce pouze pošle data na server. Obrázek aplikace je uveden v příloze na straně 35 pod číslem 13.

4.3.2.4 Správa databáze

Aplikace umožňuje administrátorovi po přihlášení spravovat databázi přes administrační rozhraní Django. Toto rozhraní je generováno automaticky. Rozhraní pro přístup k databázi je dostupné na adrese „*http://127.0.0.1:8000/admin*“, kde „*http://127.0.0.1:8000*“ představuje adresu serveru. Náhled do administrátorského rozhraní je zobrazen na obrázku 9.



Obrázek 9: Pohled na tabulku „Dataset“ z administračního rozhraní

4.4 Testování

Po dokončení implementace bylo nutné otestovat jednotlivé navržené funkce. Jelikož se vychází z vodopádového modelu se zpětnou vazbou, bylo možno vrátit se zpět k návrhu a objevené chyby navrhnout a implementovat znovu.

V této fázi byla objevena chyba navržení databáze. Kde byly původně navrhnuty tři tabulky. Tabulka „Park“ pro ukládání dat generovaným uživatelem z aplikace pro editaci obsazenosti parkoviště. Tabulka „Dataset“ pro výsledky porovnání těchto uživatelem

ručně zadaných dat s daty z nahraných souborů s výstupy z algoritmů pro digitální zpracování obrazu. A tabulka „Video“ měla uchovávat informace o jednotlivých videích v databázi. Tabulka „Video“ měla sloužit jako kořenová, takže tabulky „Park“ a „Dataset“ by obsahovaly primární klíč tabulky „Video“. Ale při vymazání záznamu z tabulky „Video“, vznikaly prázdné atributy v ostatních tabulkách, což bylo nežádoucí. Toto vedlo k odstranění vzájemných vztahů a vytvoření redundance, jelikož se v každé tabulce vyskytoval název videa. Poté byla odstraněna celá tabulka „Video“, jelikož již byla zbytečná.

Další objevený nedostatek byl na straně klientské aplikace. Zde implementovaný posuvník pro volbu snímku vyvolával událost „onchange“, která volala JavaScriptovou funkci „getImg“. Tato funkce volá požadavek na snímek na server. Toto vedlo při přesunu posuvníku k zahlcení serveru žádostmi o snímek (např. ze snímku číslo 1 na snímek číslo 1000 bylo na server zasláno 1000 požadavků na snímek). Tato událost byla změněna na „onmouseup“, která je vyvolávána pro uvolnění tlačítka myši nad objektem. Tímto se značně omezil počet žádostí na server a snížilo se zatížení serveru.

Pro zrychlení aplikace byla doimplementována možnost volby kvality snímku. Při nastavení vysoké kvality se z videa extrahuje snímek v rozlišení 1920x1080 pixelů. Pracovat s takto vysokým rozlišením je náročné na výpočet pro narovnání obrazu pomocí programu „Korekce perspektivní projekce“. Proto je umožněno vysokou kvalitu vypnout pomocí jednoduchého prvku „checkbox“. Při takovémto nastavení se snímek z videa exportuje v rozlišení 1366x768 pixelů. Výsledkem je znatelné zrychlení zpracování požadavku na snímek z původního průměru 1,8 sekundy na 0,8 sekundy.

5 Závěr

Cílem práce bylo navrhnout a naimplementovat aplikaci pro editaci obsazenosti parkoviště, dále navrhnout a naimplementovat aplikaci pro příjem dat z algoritmu zpracování digitálního obrazu. Po přečtení této práce byl čtenář seznámen s hlavními technologiemi využitými při vývoji těchto aplikací. Hlavní část práce se zabývala postupem vývoje aplikací. Byly zde popsány jednotlivé fáze vývoje software jako je analýza, návrh, implementace a testování.

Samotné aplikace byly vyvíjeny s důrazem na užitečnost a jednoduchost se zaměřením na cílovou skupinu uživatelů, kterou představují vývojáři algoritmů pro zpracování a analýzu digitálního obrazu. Proto je aplikace oproštěna o složité grafické efekty, které by práci s touto aplikací mohly znepřehlednit a tím znepříjemnit.

Pomocí těchto aplikací bude možné otestovat řešení vzniklé na této univerzitě, konkrétně „Detecting Free/Oocupied Places in Parking Lots“. Toto řešení je součástí většího projektu, vzniklého pro uznávané výrobce automobilů. Cílem tohoto řešení bylo vyvinout algoritmy pro automatickou detekci obsazených a volných míst parkoviště.

6 Literatura

[1] SCHARSTEIN, D. and SZELISKI, R. *A taxonomy and evaluation of dense two-frame stereo correspondence algorithms*, International Journal of Computer Vision [online]. [cit. 2013-04-18]. Dostupné z: http://vision.middlebury.edu/stereo/

[2] GEIGER, Andreas and LENZ, Philip and URTASUN, Raquel, *Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite*, Computer Vision and Pattern Recognition (CVPR) [online]. [cit. 2013-04-18]. Dostupné z: http://www.cvlibs.net/datasets/kitti/

[3] ŠVEC, Jan, *Seriál Python - Létající cirkus*, Root.cz: informace nejen ze světa Linuxu [online]. 2003 [cit. 2013-02-23]. Dostupné z: http://www.root.cz/clanky/letajici-cirkus/

[4] HOLOVATY, Adrian a Jacob KAPLAN-MOSS, *The Django Book*. [online]. 2009 [cit. 2013-05-02]. Dostupné z: http://www.djangobook.com/

[5] *Django project: Django Česká republika*, [online]. 2005–2013 [cit. 2013-05-02]. Dostupné z: http://www.djangoproject.cz/

[6] *Django: Django documentation* , [online]. 2005–2013 [cit. 2013-05-02]. Dostupné z: https://docs.djangoproject.com/

[7] *FFmpeg: FFmpeg documentation*, [online]. 2005–2013 [cit. 2013-05-02]. Dostupné z: http://www.ffmpeg.org/ffmpeg.html

[8] *W3schools.com: AJAX Introduction* , [online]. 1999–2013 [cit. 2013-05-02]. Dostupné z: http://www.w3schools.com/ajax/ajax_intro.asp

[9] GARRETT, Jesse James, *Ajax: A New Approach to Web Applications*, Adaptive path [online]. 2005 [cit. 2013-05-02]. Dostupné z: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications

[10] ZAKAS, Nicolas C. *Professional JavaScript for Web Developers*. Indianapolis, Indiana: Wiley Publishing Inc., 2005. ISBN 0470227818.

[11] FLOWLER, Martin, *Catalog of Patterns of Enterprise Application Architecture*. [online]. 2003 [cit. 2013-05-02]. Dostupné z: http://www.martinfowler.com/eaaCatalog/

[12] FLOWLER, Martin, *Patterns of Enterprise Application Architecture*. Addison Wesley, 2002. ISBN 0-321-12742-0.

[13] GAURA, Jan, *Odstranění geometrických zkreslení obrazu*, [online]. [cit. 2013-04-20]. Dostupné z: http://mrl.cs.vsb.cz/people/gaura/dzo/geom_distortion_cz.pdf

[14] *Geometric Image Transformations*, [online]. 2009 [cit. 2013-05-02]. Dostupné z: http://opencv.willowgarage.com/documentation/geometric_image_transformations.html

A Tabulky

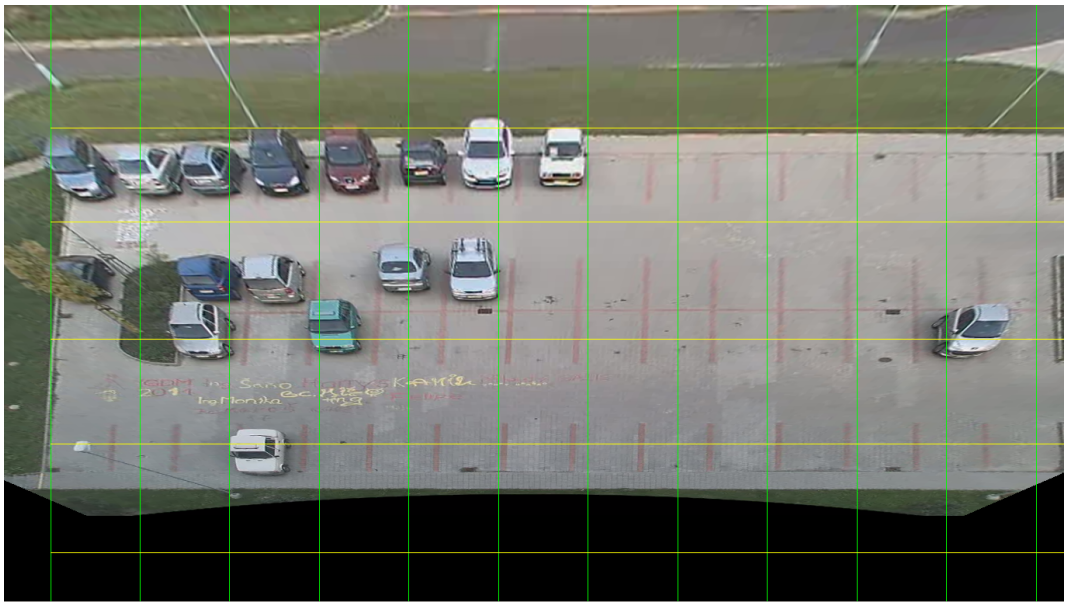
Park				
Název	**Typ**	**Délka**	**PK**	**Popis**
id	int	-	ano	primární klíč
video	char	100	ne	název videa
screen	int	-	ne	číslo snímku
p1	bool	-	ne	místo na parkovišti
p2	bool	-	ne	místo na parkovišti
p3	bool	-	ne	místo na parkovišti
p4	bool	-	ne	místo na parkovišti
p5	bool	-	ne	místo na parkovišti
p6	bool	-	ne	místo na parkovišti
p7	bool	-	ne	místo na parkovišti
p8	bool	-	ne	místo na parkovišti
p9	bool	-	ne	místo na parkovišti
p10	bool	-	ne	místo na parkovišti
p11	bool	-	ne	místo na parkovišti
p12	bool	-	ne	místo na parkovišti
p13	bool	-	ne	místo na parkovišti
p14	bool	-	ne	místo na parkovišti
p15	bool	-	ne	místo na parkovišti
p16	bool	-	ne	místo na parkovišti
p17	bool	-	ne	místo na parkovišti
p18	bool	-	ne	místo na parkovišti
p19	bool	-	ne	místo na parkovišti
p20	bool	-	ne	místo na parkovišti
p21	bool	-	ne	místo na parkovišti
p22	bool	-	ne	místo na parkovišti
p23	bool	-	ne	místo na parkovišti
p24	bool	-	ne	místo na parkovišti
p25	bool	-	ne	místo na parkovišti
p26	bool	-	ne	místo na parkovišti
p27	bool	-	ne	místo na parkovišti
p28	bool	-	ne	místo na parkovišti
p29	bool	-	ne	místo na parkovišti
p30	bool	-	ne	místo na parkovišti
p31	bool	-	ne	místo na parkovišti
p32	bool	-	ne	místo na parkovišti
p33	bool	-	ne	místo na parkovišti
p34	bool	-	ne	místo na parkovišti
p35	bool	-	ne	místo na parkovišti

p36	bool	-	ne	místo na parkovišti
p37	bool	-	ne	místo na parkovišti
p38	bool	-	ne	místo na parkovišti
p39	bool	-	ne	místo na parkovišti
p40	bool	-	ne	místo na parkovišti
p41	bool	-	ne	místo na parkovišti
p42	bool	-	ne	místo na parkovišti
p43	bool	-	ne	místo na parkovišti
p44	bool	-	ne	místo na parkovišti
p45	bool	-	ne	místo na parkovišti
p46	bool	-	ne	místo na parkovišti
p47	bool	-	ne	místo na parkovišti
p48	bool	-	ne	místo na parkovišti
p49	bool	-	ne	místo na parkovišti
p50	bool	-	ne	místo na parkovišti
p51	bool	-	ne	místo na parkovišti
p52	bool	-	ne	místo na parkovišti
p53	bool	-	ne	místo na parkovišti
p54	bool	-	ne	místo na parkovišti
p55	bool	-	ne	místo na parkovišti
p56	bool	-	ne	místo na parkovišti
p57	bool	-	ne	místo na parkovišti
p58	bool	-	ne	místo na parkovišti

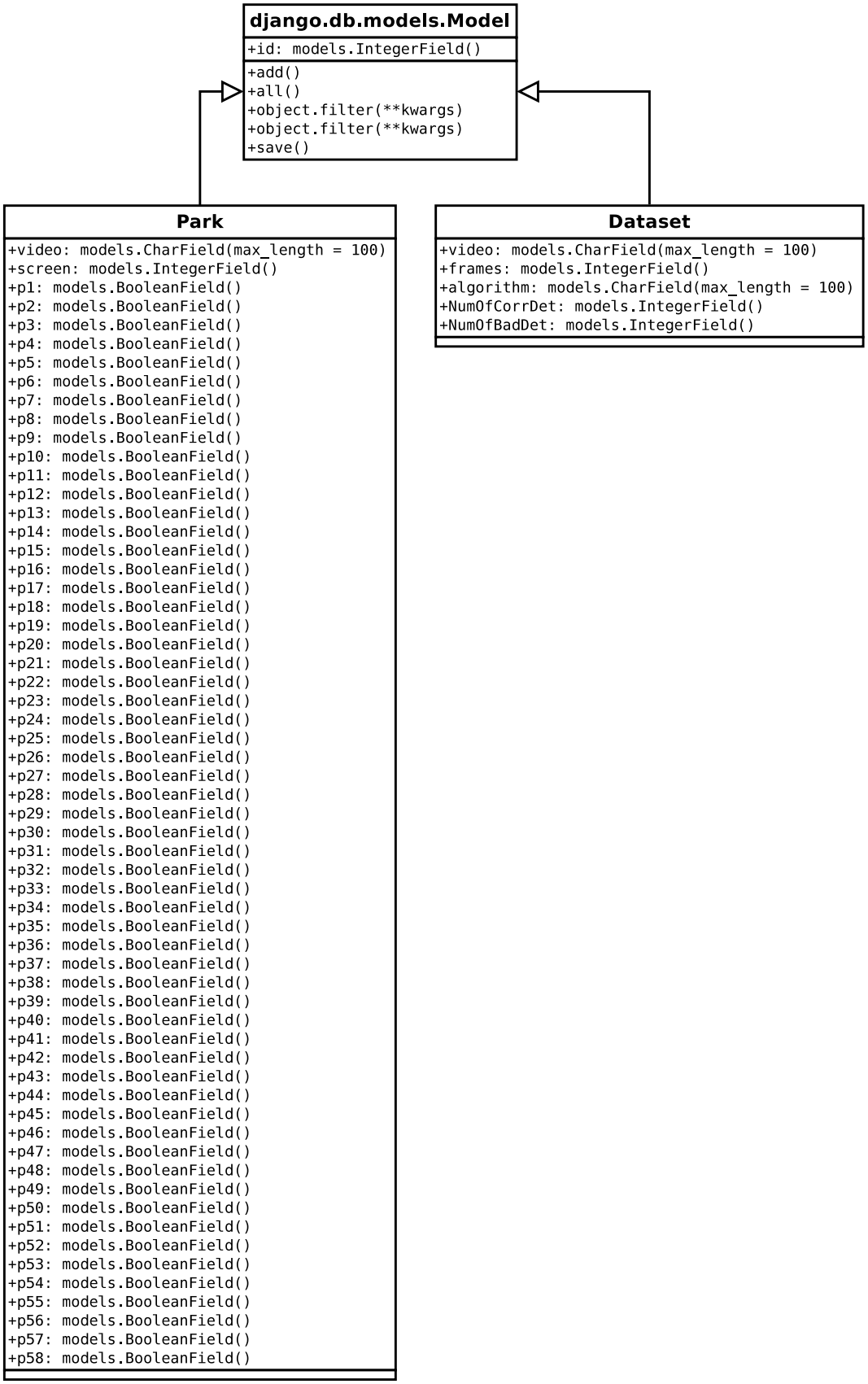
Tabulka 2: Datový slovník - Park

B Obrázky

Obrázek 10: Snímek po odstranění distorze



Obrázek 11: Snímek po narovnání

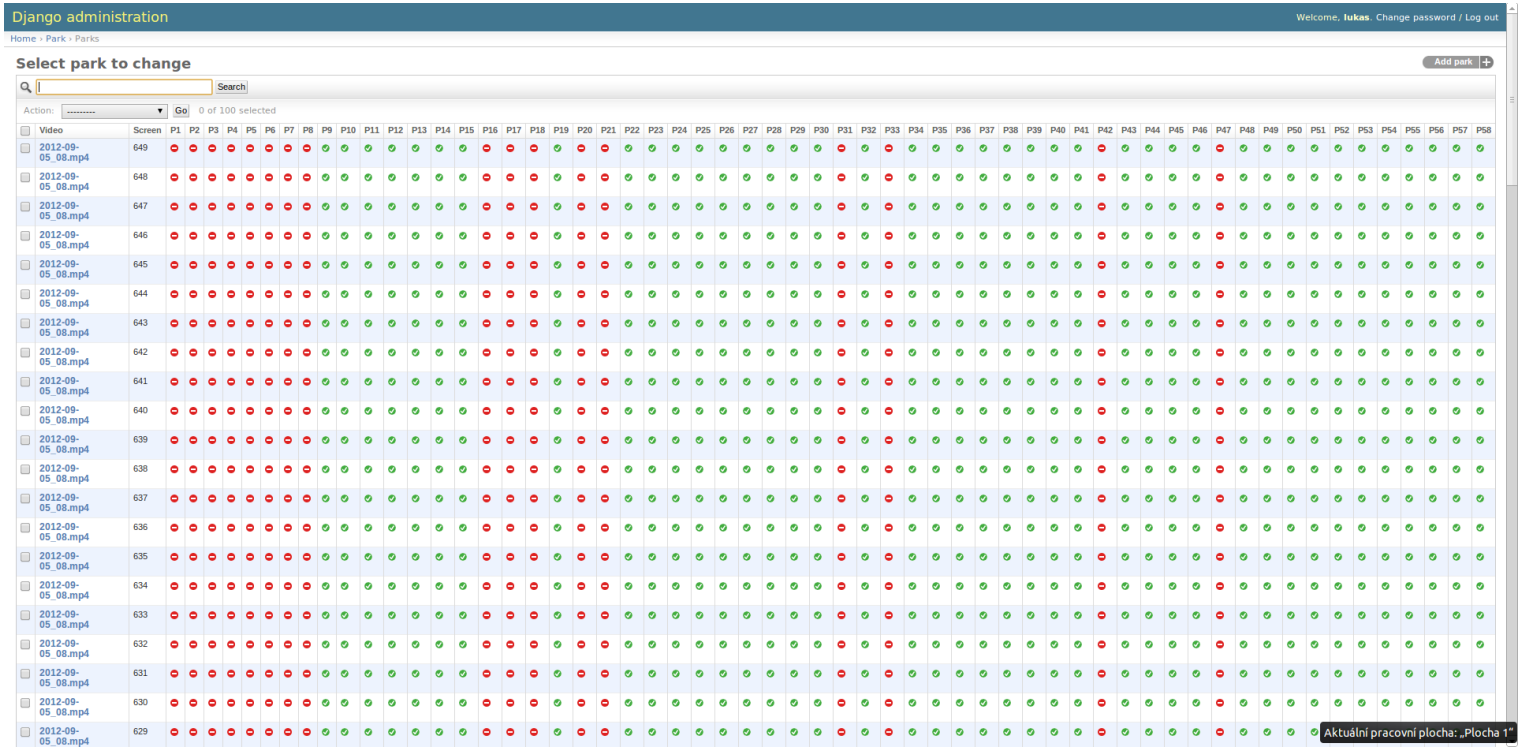


Obrázek 12: Třídní diagram reprezentující databázové schéma

dataset

Jmeno videa	Pocet snimku	Algoritmus	Pocet spravnych detekci	Pocet spatnych detekci
2012-09-05_08.mp4	649	XML	26158	11484
2012-09-05_08.mp4	649	CSV	26158	11484
2012-09-05_08.mp4	649	CSV 2	26156	11486
2012-09-05_08.mp4	649	XML bez chyby	37642	0
2012-09-05_08.mp4	649	CSV bez chyby	37642	0

Obrázek 13: Aplikace pro příjem dat z algoritmu zpracování obrazu



Obrázek 14: Pohled na tabulku „Park“ z administračního rozhraní

C Implementace

Součástí práce je také CD s implementací a elektronickou podobou práce ve formátu PDF.

C.1 Instalace

Pro zprovoznění práce je potřeba nainstalovat Djnago framework. Postup pro zprovoznění je napsaný pro práci z terminálu na platformě Linux Ubuntu/Debian.

1. z oficiálních stránek projektu je potřeba stáhnout Django framework *„https://www.djangoproject.com/download/“* tento balík je také přiložen na CD
2. pomocí následujících příkazů se Django framework rozbalí a nainstaluje:
   - *tar xzvf Django-1.5.1.tar.gz*
   - *cd Django-1.5.1*
   - *sudo python setup.py install*
3. dále je potřeba doinstalovat FFmpeg příkazem *sudo apt-get install ffmpeg*
4. po instalaci se přesuneme do adresáře s bakalářskou prací konkrétně */implementace/django/Parkoviste*
5. příkazem *python manage.py runserver* se spustí Django server
6. aplikace pro editaci obsazenosti parkoviště se nachází v adresáři */implementace/web* a je ji možné spustit souborem *aplikace.html*
7. aplikace pro příjem dat z algoritmů zpracování obrazu se nachází v adresáři */implementace/web_dataset* a je ji možné spustit souborem *dataset.html*